

UNIwersytet WarMińsko-Mazurski w Olsztynie
Wydział Matematyki i Informatyki

Kierunek: Informatyka

Igor Sachnowski

**Internetowy system przeglądania wiadomości na
podstawie preferencji użytkownika.**

Praca inżynierska wykonana
w katedrze Metod Informatycznych
pod kierunkiem
dr Krzysztofa Sopyły

Olsztyn, 2016 rok

UNIVERSITY OF WARMIA AND MAZURY IN OLSZTYN
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Field of Study: Computer Science

Igor Sachnowski

**IT online system for browsing news based on user
preferences.**

Engineer's Thesis is performed
in the Department of Multimedia and Computer Graphics
under supervision of
dr Krzysztof Sopyła

Olsztyn, 2016

Spis treści

Streszczenie	3
Summary	4
Rozdział 1. Wstęp	5
1.1. Układ pracy	6
Rozdział 2. Opis technologii wykorzystanych do implementacji aplikacji	7
2.1. Środowisko Meteor oraz jego komponenty.	7
2.1.1. Meteor	7
2.1.2. JavaScript	9
2.1.3. MongoDB	10
2.2. Język programowania Python	11
2.3. Techniki sztucznej inteligencji	12
2.3.1. SVM	12
2.3.2. Bag of words	13
Rozdział 3. Projekt techniczny systemu informatycznego	15
3.1. Diagram przypadków użycia	15
3.2. Scenariusze	16
3.2.1. Dodaj nową domenę	16
3.2.2. Zmień preferencji wyświetlania	16
3.2.3. Przeglądaj wiadomości	17
3.2.4. Zbierz dane o witrynie	17
3.2.5. Utwórz słownik	18
3.2.6. Zbuduj reprezentację Bag of Words	19
3.2.7. Zbierz podstrony	20
3.2.8. Trenuj model SVM	20
3.2.9. Nadaj kategorię dla strony	21
Rozdział 4. Implementacja aplikacji	23
4.1. Architektura aplikacji - diagram komponentów	23
4.2. Sposób działania - diagram czynności	23
4.3. Schemat bazy danych	25
4.4. Implementacje	27
4.4.1. Implementacja Crawlera WWW	27
4.4.2. Implementacja tworzenia słownika	29
4.4.3. Implementacja tworzenia wektorów	30
4.4.4. Implementacja budowania modelu SVM	31
4.4.5. Implementacja menadżera	32
4.5. Interfejs użytkownika	33
4.5.1. Strona główna	33
4.5.2. Kategorie	34
4.5.3. Dodaj stronę	35

4.6.	Instrukcja instalacji	36
4.6.1.	Instalacja Node.js	36
4.6.2.	Instalacja Meteora	36
4.6.3.	Uruchamianie aplikacji	36
Rozdział 5.	Podsumowanie	37
Bibliografia		38
Spis rysunków		39
Listings		40

Streszczenie

W dobie dostępu do ogromnej ilości informacji w postaci tekstowej, coraz większą rolę odgrywają metody i techniki pozwalające na dostosowanie treści do naszych preferencji. Celem pracy jest stworzenie systemu wspomagania zbierania, filtrowania i zagregowanego prezentowania wiadomości z sieci (blogi, strony tematyczne) na podstawie preferencji użytkownika. Dzięki utworzonemu rozwiązaniu, użytkownik sam decyduje z jakich dziedzin chciałby otrzymywać artykuły bądź wiadomości, które system sam klasyfikuje i przydziela do odpowiednich kategorii.

W ramach pracy wykorzystano techniki sztucznej inteligencji: SVM oraz Bag of words do budowania modelu, który posłużył do automatycznego przydzielania tekstów do kategorii. Kolejnym wyzwaniem w pracy był mechanizm składowania ogromnych ilości danych. Do rozwiązania tego problemu wykorzystano nierelacyjny system bazy danych MongoDB oraz środowisko do wysoce skalowalnych aplikacji www - Node.js. Do prezentacji treści i działania aplikacji wykorzystano środowisko Meteor, które jest platformą pozwalającą na tworzenie aplikacji czasu rzeczywistego. Polega to na wymianie danych między klientem, a serwerem na bieżąco. Aby dostać nowe wiadomości z bazy danych użytkownik nie potrzebuje przeładowywać strony, ponieważ działające w tle połączenie asynchroniczne z bazą danych sprawia, że dostaje je na bieżąco.

Summary

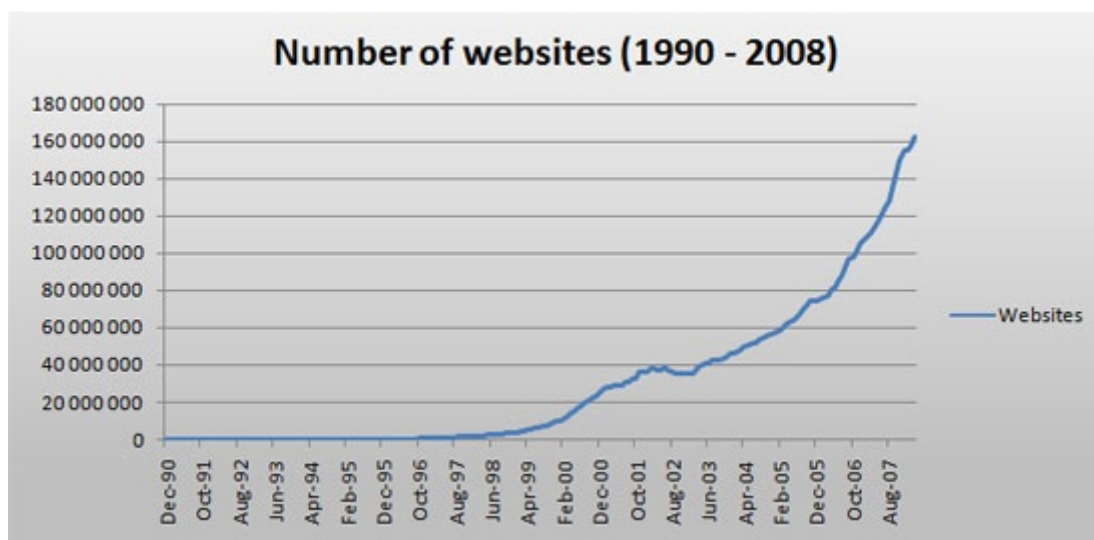
In the age of access to fast amounts of information in text form, increasing the role of methods and techniques for adapting content to our preferences. Purpose of the thesis is create support system to collect, filter and aggregated presentation of informations from web (blogs, thematic sites) based on user preferences. The created solution, the user decides he would like to received articles or informations, that the system classifies and assign to appropriate category.

Within the case artificial intelligence techniques has been used: **SVM** and **Bag of words** to create model, which help to auto-assign text to categories. The next challenge in the case was mechanism for the storage of the huge amounts of data. To resolve this issue author used non-relational system of database MongoDB and framework to highly scalable web applications - Node.js. For the presentation of the content and application uses framework Meteor, which is a platform that allows to build real-time web applications. It's about to exchange of data between the client and the server in the real time. To receive new informations from database, user dont need to refresh site, because running in the background asynchronous connection to the database gets them on regular basis.

Rozdział 1

Wstęp

W czasach, gdy użytkownik jest zalewany sporą ilością informacji posiadając jednocześnie niewiele czasu, uciążliwe jest odszukanie interesujących nas wiadomości. Od parunastu lat internet rozwinął się do niewiarygodnych rozmiarów (rys. 1.1), a także stał się łatwiej dostępny dla przeciętnego człowieka. Sporo osób także zaczęło korzystać z prasy internetowej, zamiast z jej papierowego odpowiednika. Spowodowało to wzrost ilości portali informacyjnych, blogów czy serwisów oferujących wiadomości w interesującej nas dziedzinie.



Rysunek 1.1. Liczba stron internetowych na przestrzeni lat.

[źródło: http://farm4.static.flickr.com/3095/2387270804_c8a14cc063_o.jpg]

Celem pracy było stworzenie aplikacji internetowej, która zaoszczędzi czas użytkownika na przeglądanie swoich ulubionych portali informacyjnych i pozwoli mu pominąć proces wyszukiwania interesujących wiadomości. Utworzony system pozwoli przeciętnemu użytkownikowi skupić interesujące go wiadomości w jednym miejscu.

Aplikacja sama wyświetla kolejne informacje z ulubionych kategorii użytkownika i na podstawie jego ocen dobiera mu kolejne wiadomości z bazy danych. Poprzez algorytm oceniania użytkownik decyduje, które wiadomości go interesują, a które chciałby pomijać i nie oglądać ich w przyszłości. Użytkownik sam może dodawać nowe witryny do bazy danych. Zostają one pobrane wraz z zawartością i są weryfikowane przez algorytm kategoryzacji oparty na klasyfikatorze SVM oraz metodzie Bag of words.

1.1. Układ pracy

Praca została podzielona na 5 różnych sekcji. W każdej z nich poruszone są inne zagadnienie dotyczące tejże pracy.

W pierwszym rozdziale opisany jest wstęp do pracy. W tej części nakreślony jest problem poruszony przez autora pracy oraz zaproponowane rozwiązanie problemu.

Drugi rozdział przedstawia technologie, które zostały wykorzystane do realizacji projektu. Są to m. in. środowisko Meteor, język Python czy system bazy danych MongoDB.

Trzeci rozdział poświęcony jest pokazaniu całej aplikacji w postaci diagramów UML i scenariuszy symulujących przebieg zdarzeń. W tym dziale zostało szczegółowo opisane działanie aplikacji oraz przypadki wykorzystywania aplikacji.

W rozdziale czwartym przedstawiony jest sposób implementacji systemu, poszczególne skrypty, a także fragmenty interfejsu użytkownika. W tej części zaprezentowano wybrane fragmenty kodu aplikacji oraz część systemu, która jest widoczna dla zalogowanego użytkownika.

W piątej części zawarte jest podsumowanie całej pracy.

Rozdział 2

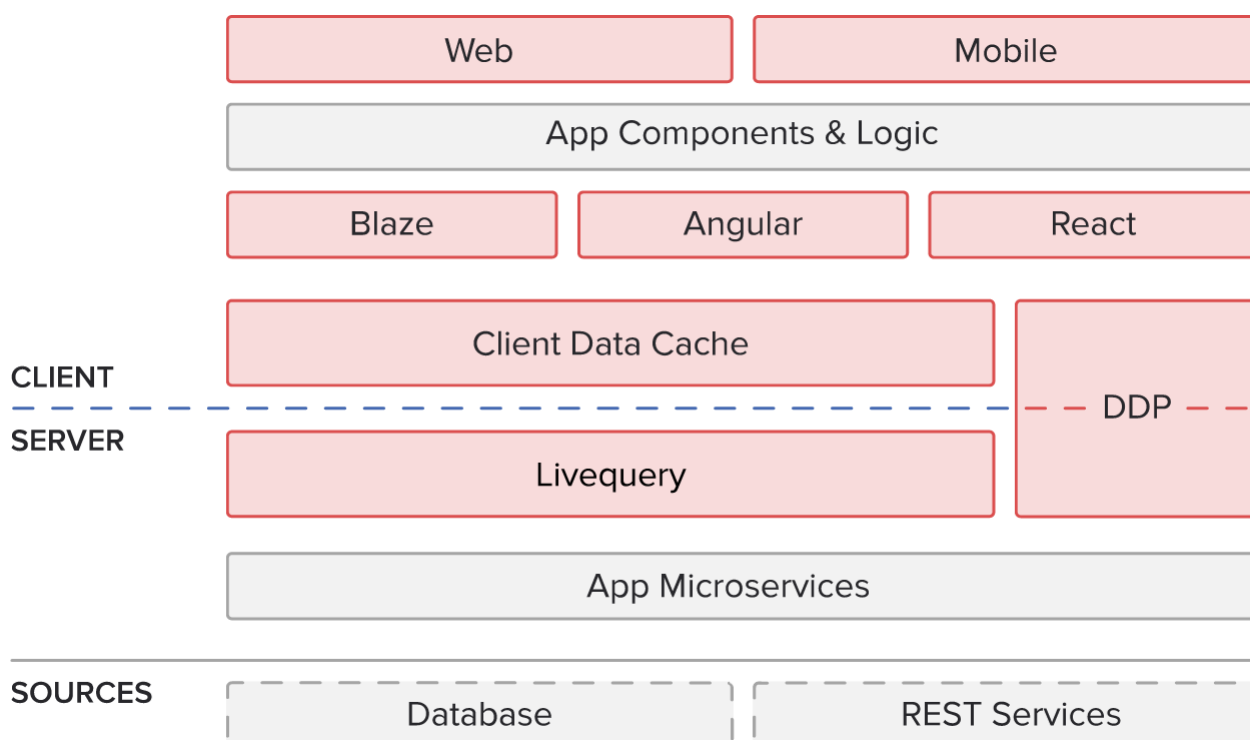
Opis technologii wykorzystanych do implementacji aplikacji

2.1. Środowisko Meteor oraz jego komponenty.

W tej sekcji zostało omówione środowisko Meteor, język skryptowy JavaScript na którym opiera się Meteor oraz baza danych MongoDB wykorzystana przy całym projekcie.

2.1.1. Meteor

Meteor jest to darmowa platforma, na bazie NodeJS, do tworzenia stron internetowych i mobilnych działających w czasie rzeczywistym. Jest to technologia, która umożliwia użytkownikowi otrzymywanie informacji na bieżąco od razu po dodaniu ich do bazy danych. Platforma Meteor opiera się na języku Java Script zarówno po stronie klienta jak i serwera, oraz współpracuje z bibliotekami tego języka np. AngularJS czy ReactJS.



Rysunek 2.1. Struktura danych Meteor'a.

[źródło: https://d14xs1qewsqjcd.cloudfront.net/assets/i/technology/meteor_framework.svg]

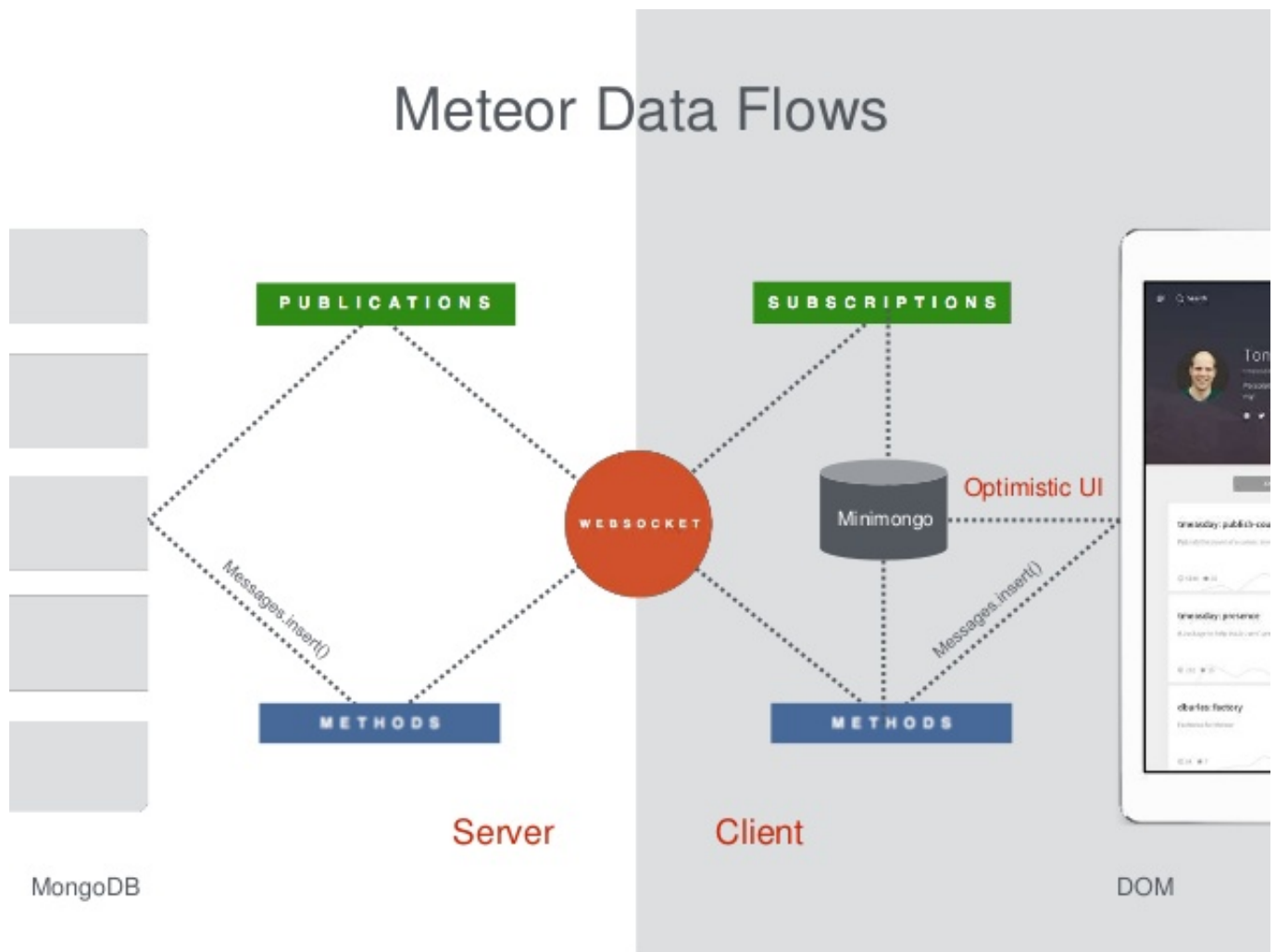
Meteor został zaprezentowany po raz pierwszy w 2011 roku pod nazwą Skybreak jako platforma do tworzenia stron internetowych. W 2012 roku nazwa została zmieniona na Meteor. Aplikacja była pierwotnie możliwa do uruchomienia tylko pod systemami UNIX, od wersji 1.1 Meteor jest wspierany przez system Windows[5].

Wyjątkowość Meteora stanowiła realizacja działania aplikacji w czasie rzeczywistym zarówno po stronie klienta jak i serwera poprzez DDP, czyli Distributed Data Protocol. Jest to protokół utworzony przez autorów platformy Meteor do synchronizacji danych pomiędzy serwerem a klientem. Z racji wspierania tych mechanizmów, jako źródło danych wykorzystano system bazy danych MongoDB. Podczas działania aplikacji wszystkie dane przechowywane są w MongoDB na serwerze, a po stronie klienta tworzone jest MiniMongo. Jest to implementacja kolekcji z bazy danych znajdująca się po stronie klienta. Jest zbiorem danych znajdującym się w pamięci operacyjnej klienta, z API stworzonym na podobieństwo tego z Mongo. MiniMongo zachowuje się jak emulator naszej bazy danych po stronie serwera, czyli MongoDB, na którym możemy operować używając interfejsu użytkownika. Połączenie między MongoDB, a MiniMongo jest cały czas utrzymywane na websocket'ach. (rys. 2.2)

Meteor jest platformą, która stawia przede wszystkim na prostotę, wymaga od programisty głównie znajomości JavaScript. Platforma ta, wymaga określonej struktury plików. Pliki w folderze `server` są uruchamiane po stronie serwera, w folderze `client` - po stronie klienta, w pozostałych przypadkach zaś, pliki są uruchamiane po obu stronach. W strukturze występuje także folder `lib`, w którym pliki są uruchamiane jako pierwsze.

Do platformy Meteor jest stworzona spora baza pakietów, które wspomagają i przyspieszają pracę nad aplikacją. Są to paczki między innymi do tworzenia aplikacji mobilnych (np. Cordova), lokalizacji GPS (np. GPS-Tracking) czy wyglądu strony (np. bootstrap-3). Pakiety te dzielą się na:

- Rdzenne pakiety Meteor'a, które zawierają podstawowe paczki niezbędne do funkcjonowania środowiska.
- Pakiety smart, to pakiety dostarczane razem z Meteor'em i można je w każdej chwili dodać do projektu jedną komendą.
- Pakiety lokalne, są to moduły utworzone przez użytkownika. Powinny one znajdować się w folderze `packages` w projekcie.
- Pakiety smart Atmosphere - są to paczki firm trzecich, których dystrybucją zajmuje się serwis Atmosphere.js[2] .
- Pakiety NPM - paczki do Node.js, które nie współpracują bezpośrednio z Meteor'em ale wspomagają prace jego pakietów.



Rysunek 2.2. Przepływ danych w środowisku Meteor.

[źródło:

<http://image.slidesharecdn.com/meteor-rails-2015-rev1-150710182721-lva1-app6891/95/meteor-rails2015-16-638.jpg?cb=1436553027>

2.1.2. JavaScript

JavaScript jest skryptowym językiem programowania przeznaczonym głównie do pisania stron www. Język został utworzony przez firmę Netscape w 1995 roku oraz miał głównie na celu zarządzanie i przekształcanie treści stron internetowych. Skrypty stworzone w tym języku służą do utworzenia interaktywności na stronach internetowych np. obsługa formularzy czy wydarzeń. Obecnie kod wykonywany jest po stronie klienta (przeglądarka) i serwera (Node.js).

Podstawą JavaScriptu jest ECMAScript[1], czyli język skryptowy, który powstał poprzez standaryzację JavaScriptu. Standaryzacja powoduje to, że niezależnie od przeglądarki jakiej używa użytkownik, wszystkie funkcje oraz ich dane będą zachowywały się tak samo. Ponadto można implementować obiekty, właściwości, typy i wartości, których nie zawiera standard ECMA-262. Najpopularniejsze z nich to XMLHttpRequest (wtyczka służąca do komunikacji z serwerem w tle) i ActiveXObject (środowisko do tworzenia obiektów ActiveX).

JavaScript oferuje szereg dostępnych bibliotek. Najpopularniejsze z nich to:

- jQuery - biblioteka, która umożliwia działanie w zakresie tworzenia efektów w postaci animacji, rozbudowy obsługi zdarzeń, manipulowania modelem DOM czy obsługę zapytań AJAX. jQuery wyróżnia przede wszystkim elastyczność przy dostosowywaniu się do przeglądarki, wygoda przy tworzeniu wtyczek a także niewielkie rozmiary.
- Angular.js - otwarty projekt firmy Google, który miał za zadanie wprowadzić wzorca MVC[4] do aplikacji www, aby ułatwić ich testowanie i rozwój. Angular miał przede wszystkim oddzielać warstwę klienta od warstwy serwerowej przy tworzeniu aplikacji, co miało wpływ na niezależną pracę obu tych środowisk.
- Node.js - jest otwartym środowiskiem programistycznym, opartym na silniku V8 autorstwa Google. Pozwala na tworzenie wysoce skalowalnych aplikacji www używając przy tym asynchronicznego systemu wejścia-wyjścia.

2.1.3. MongoDB

MongoDB jest nierelacyjną bazą danych. Cechuje ją duża wydajność, skalowanie się oraz brak określonej struktury całej bazy. Cały system został napisany w oparciu o język C++, a dane zapisywanie są w dokumentach formatu BSON w oparciu o kolekcje.

MongoDB jest systemem opartym o ruch NoSQL (Not only SQL). Jest to ruch który powstał jako alternatywa dla relacyjnych systemów baz danych. Dzięki odrzuceniu relacyjnego modelu bazy, dane nie muszą być przechowywane według ściśle określonych reguł, stają się skalowalne i dzięki temu szybciej i efektywniej obsługują zapytania. Według tej teorii, problemy ze skalowaniem, mają być rozwiązane na poziomie serwera. Do ruchu NoSQL zaliczają się też takie systemy baz danych jak CassandraDB, CouchDB, Accumulo czy Aerospike.

Pierwsza stabilna wersja MongoDB została wydana w 2009 roku oznaczona numerem wersji 0.9. Od tamtej pory wydano 11 stabilnych wersji, a najnowszą jest 3.2. MongoDB rozwinęło się do dojrzałego systemu, który został użyty w dużych projektach jak serwis popularnego, amerykańskiego dziennika *The New York Times* czy projekt Wielkiego Zderzacza Hadronów.

Dane w MongoDB są przechowywane w formacie BSON, który jest w większości oparty o format JSON. Są one zapisywane w formie binarnej stąd nazwa formatu który oznacza **Binary JSON**. Tabele nie są ze sobą powiązane, a do każdego rekordu wstawionego do bazy danych przyporządkowany jest automatycznie generowany unikalny identyfikator („_id”). Baza danych jest bazą dynamiczną co sprawia, że dane przy dodawaniu rekordów muszą być dokładnie sprawdzone przed wprowadzaniem tj. nazwa kolumny, kolekcji i bazy, ponieważ w razie pomyłki omyłkowo zostanie utworzona nowa kolumna lub tabela.

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
    phone: "123-456-7890",
    email: "xyz@example.com"
  },
  access: {
    level: 5,
    group: "dev"
  }
}
```

Rysunek 2.3. Przykładowy rekord w MongoDB

2.2. Język programowania Python

Część „back-end’owa” systemu została wykonana w języku Python. Jest on obiekowym językiem programowania wysokiego poziomu. Jest to język dostępny na wszystkich platformach. Język ten wymusza używanie wcięć do określenia bloków programu, zamiast klamr. Ze względu na skrupulatnie określoną konstrukcję kodu, w którym białe znaki odgrywają główną rolę, programy są przejrzyste i czytelne. Python oferuje możliwość pisania obiektowego, strukturalnego czy funkcyjnego. Pamięć jest alokowana automatycznie, a do zarządzania nią stosuje się tzw. Garbage Collector, który sprawdza, które są nieaktywne (żadna ze zmiennych nie przechowuje referencji do nich) i automatycznie je zwalnia. Python jest językiem dynamicznym, oznacza to że w kodzie programu nie trzeba określać typu danych, program sam je ustala w czasie wykonywania na podstawie wnioskowania.

Język oferuje szereg modułów i bibliotek do pisania aplikacji, które ułatwiają tworzenie i komponowanie aplikacji. Są to rozbudowane moduły takie jak:

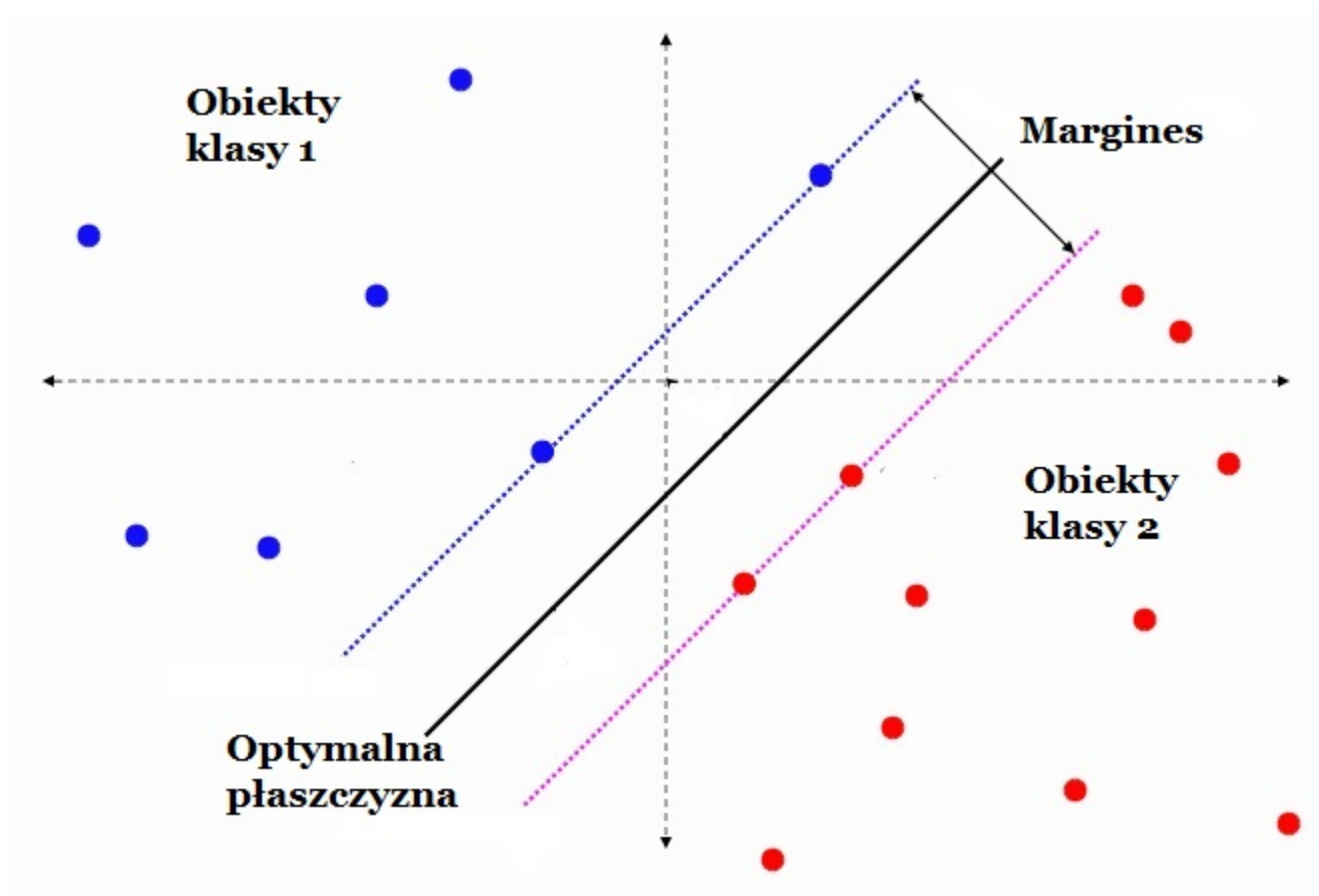
- Django, który jest środowiskiem służącym do programowania aplikacji www. Posiada on świetną dokumentację, automatyczny panel admina oraz podział kodu na szablony, modele, formularze i widoki.
- Pandas - środowisko do analizowania danych i statystycznych obliczeń. Charakteryzuje się dużą funkcjonalnością związaną z manipulacją danymi. Dane można w prosty sposób konwertować, przekształcać, przemieszczać, indeksować czy hierarchizować.
- PyMongo - jest rozszerzeniem służącym do wykonywania operacji na bazach MongoDB w Pythonie. Oferuje prosty sposób łączenia z bazą i proste pobieranie danych.
- scikit-learn - paczki służące do technik sztucznej inteligencji proszujące tzw. uczenie maszynowe (ang. machine learning). Zawiera m.in. implementacje algorytmu SVM.
- urllib2 - jest to moduł, który służy do pobierania zasobów stron internetowych za pomocą adresu URL witryny.

2.3. Techniki sztucznej inteligencji

W tej sekcji opisane są metody sztucznej inteligencji użyte do algorytmu określania kategorii pobieranych stron internetowych.

2.3.1. SVM

SVM (ang. Support Vector Machine) czyli Maszyna Wektorów Nośnych to metoda klasyfikacji obiektów. Algorytm na podstawie położenia obiektów wyznacza płaszczyznę, która określa granicę pomiędzy klasami danych obiektów, a kolejne obiekty klasyfikuje na podstawie ustalonej już granicy klas. Płaszczyzna jest zbiorem rozwiązań, które odpowiadają granicy dwóch klas. SVM charakteryzuje się tym, że wybiera płaszczyznę z najszerszym marginesem, ponieważ powoduje to lepszą i dokładniejszą klasyfikację oraz zapobiega radykalnym zmianom klasyfikacji.

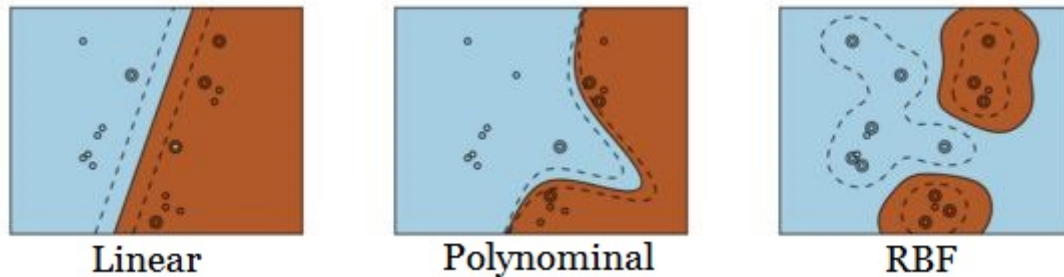


Rysunek 2.4. Przykład granicy decyzyjnej wyznaczonej przez SVM.

[źródło: <http://www.ifp.illinois.edu/~yuhuang/samsung/SVM.jpg>]

Algorytm SVM pozwala dokonywać klasyfikacji dla różnych typów granic decyzyjnych: liniowych i nieliniowych w zależności od wybranego jądra:

- Linear (liniowy) - $w * x + b = 0$
- Polynomial (nieliniowy) - $K(x, y) = (x^T * y + c)^d$
- RBF (Radial Basis Function, nieliniowy) - $K(x, x') = \exp(-\frac{\|x-x'\|^2}{2\sigma^2})$



Rysunek 2.5. Typy jąder SVM

[źródło: http://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html]

Biblioteki, które implementują algorytm SVM w języku Python to scikit-learn, LIBSVM oraz SVMstruct Python. Najpopularniejszą z nich jest scikit-learn[3], ze względu na uniwersalność, minimalizację zaalokowanej pamięci oraz prostotę użycia.

2.3.2. Bag of words

Bag of words czyli worek słów jest metodą reprezentacji wektorowej tekstów, na podstawie słów zawartych w dokumencie. Używa się jej, aby sprowadzić dane do wartości numerycznych. Ze wszystkich dokumentów budujemy słownik. Przechowuje on alfabetycznie ułożone wszystkie słowa znajdujące się w dokumentach. Na jego podstawie możemy zbudować reprezentacje wektorów dla każdego tekstu.

Klasyfikując zapisujemy liczebność wcześniej ustalonych słów w danym tekście. Jako przykład niech posłuży zdanie:

"Ala ma kota, kot ma Ale".

- W pierwszym etapie zbieramy wszystkie słowa do „worka”.
„Ala”, „ma”, „kota”, „kot”, „Ale”
- Następnie na podstawie zebranych słów, budujemy słownik
[„Ala”, „Ale”, „kota”, „kot”, „ma”]
- Dla wyżej wymienionego tekstu, na podstawie zawartości zbudowanego słownika i ich pozycji w słowniku, budujemy wektor poprzez zliczenie wystąpień danych słów.
[1,1,1,1,2]

Na podstawie tych etapów możemy tworzyć wektory i porównywać ich zawartość za pomocą iloczynu skalarnego wektorów.

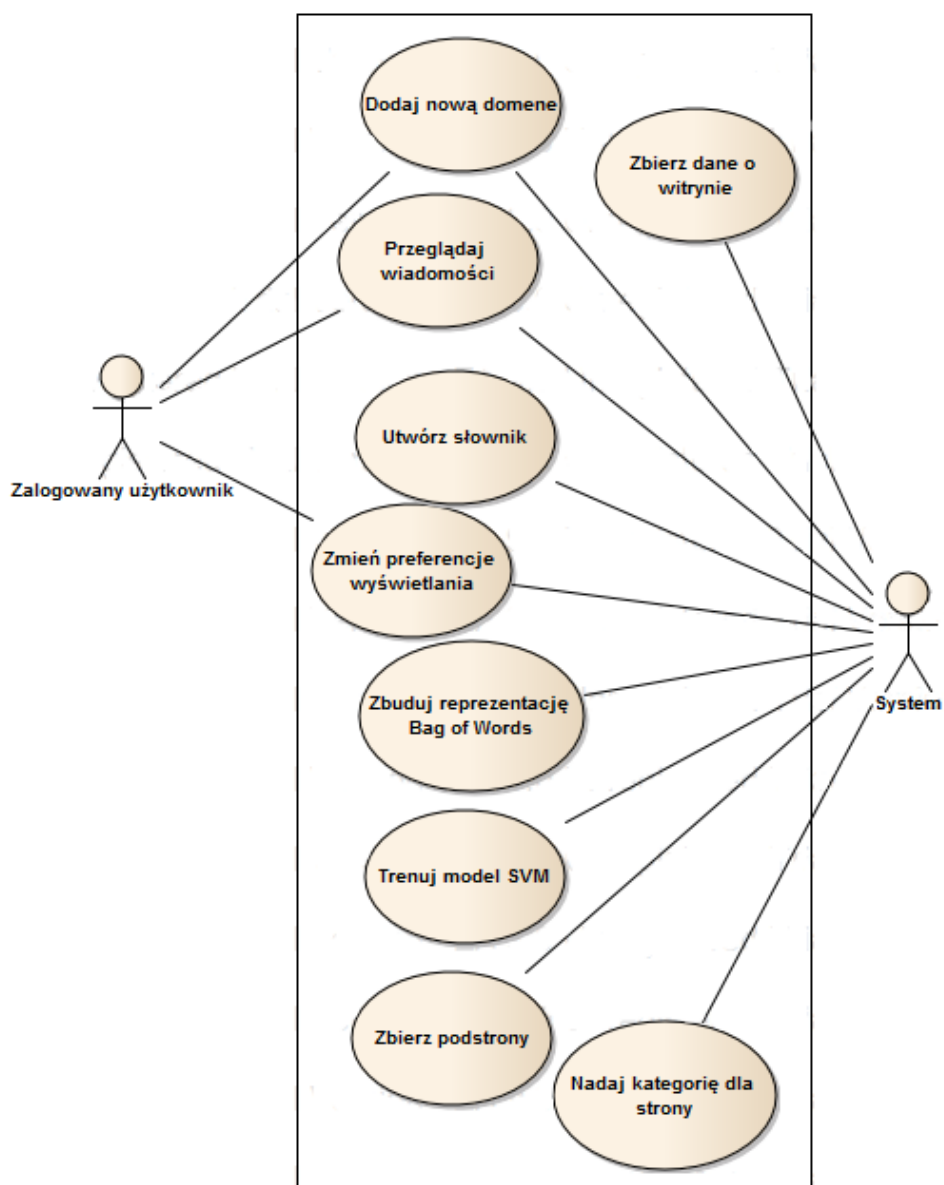
$\begin{bmatrix} 3 \\ 0 \\ 2 \\ 7 \end{bmatrix}$	$\begin{bmatrix} 3 \\ 0 \\ 3 \\ 7 \end{bmatrix}$	$\begin{bmatrix} 3 \\ 0 \\ 2 \\ 7 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 6 \\ 0 \\ 0 \end{bmatrix}$
$\begin{bmatrix} 3 & 0 & 2 & 7 \end{bmatrix}^T \cdot \begin{bmatrix} 3 \\ 0 \\ 3 \\ 7 \end{bmatrix} = 64$		$\begin{bmatrix} 3 & 0 & 2 & 7 \end{bmatrix}^T \cdot \begin{bmatrix} 0 \\ 6 \\ 0 \\ 0 \end{bmatrix} = 0$	
<p>2 podobne teksty</p>		<p>2 różne teksty</p>	

Rysunek 2.6. Porównanie dwóch wektorów utworzonych metodą Bag of words

Rozdział 3

Projekt techniczny systemu informatycznego

3.1. Diagram przypadków użycia



Rysunek 3.1. Diagram przypadków użycia

3.2. Scenariusze

W tej sekcji przedstawione są scenariusze opisujące przebieg przypadków użycia.

3.2.1. Dodaj nową domenę

Nr przypadku użycia:	1
Nazwa przypadku użycia:	Dodaj nową domenę
Aktorzy:	Zalogowany użytkownik, System
Wymaganie wstępne:	Użytkownik jest zalogowany.
Przepływ:	<ol style="list-style-type: none">1. Użytkownik lub system przekazuje adres, który ma zostać dodany do bazy.2. System łączy się z bazą danych3. System dodaje witryne do kolekcji „WebPages”.
Przepływ alternatywny:	2a. System nie może połączyć się z bazą danych i próba dodania witryny zostaje przerwana.
Warunki końcowe:	System wraca do ekranu głównego aplikacji.

Tablica 3.1. Scenariusz dla przypadku użycia: Dodaj nową domenę.

3.2.2. Zmień preferencji wyświetlania

Nr przypadku użycia:	2
Nazwa przypadku użycia:	Zmień preferencje wyświetlania
Aktorzy:	Zalogowany użytkownik, System
Wymagania wstępne:	Użytkownik jest zalogowany
Przepływ:	<ol style="list-style-type: none">1. Użytkownik otwiera aplikację na liście kategorii2. Użytkownik zaznacza interesujące go kategorie z których chciałby otrzymywać artykuły3. Użytkownik akceptuje wybór poprzez kliknięcie odpowiedniego przycisku4. System aktualizuje wybrane przez użytkownika kategorie w bazie danych
Przepływ alternatywny:	4a. System nie może połączyć się z bazą danych
Warunki końcowe	Powrót do ekranu głównego aplikacji.

Tablica 3.2. Scenariusz dla przypadku użycia: Zmień preferencje wyświetlania.

3.2.3. Przeglądaj wiadomości

Nr przypadku użycia:	3
Nazwa przypadku użycia:	Przeglądaj wiadomości
Aktorzy:	Zalogowany użytkownik, System
Wymagania wstępne:	Użytkownik jest zalogowany.
Przepływ:	<ol style="list-style-type: none">1. Użytkownik otwiera aplikację na ekranie głównym.2. System wyświetla wiadomość z dobranych przez użytkownika kategorii3. Użytkownik klikając odpowiedni przycisk przechodzi do kolejnej wiadomości.
Przepływ alternatywny:	<ol style="list-style-type: none">2a. Użytkownik nie wybrał żadnej kategorii2b. Nie ma dla użytkownika nowych wiadomości w bazie danych z wybranych kategorii.
Warunki końcowe	Wyświetlenie strony z artykułem lub komunikat o jej braku.

Tablica 3.3. Scenariusz dla przypadku użycia: Przeglądaj wiadomości.

3.2.4. Zbierz dane o witrynie

Nr przypadku użycia:	4
Nazwa przypadku użycia:	Zbierz dane o witrynie
Aktorzy:	System
Wymagania wstępne:	Poprawny adres url w bazie danych.
Przepływ:	<ol style="list-style-type: none">1. System pobiera adres witryny z bazy danych2. Pobiera całą zawartość strony łącznie z jej kodem3. System „oczyszcza” tekst strony z funkcji skryptowych i tagów HTML4. System aktualizuje dany adres url w bazie danych o zawartość strony oraz o tekst zawarty na niej.
Przepływ alternatywny:	<ol style="list-style-type: none">1a. System nie może połączyć się z bazą danych4a. System nie może połączyć się z bazą danych
Warunki końcowe	Zaktualizowana witryna w bazie danych

Tablica 3.4. Scenariusz dla przypadku użycia: Zbierz dane o witrynie.

3.2.5. Utwórz słownik

Nr przypadku użycia:	5
Nazwa przypadku użycia:	Utwórz słownik
Aktorzy:	System
Wymagania wstępne:	Witryny w bazie danych zaktualizowane o zawartość strony
Przepływ:	<ol style="list-style-type: none">1. System z każdej witryny pobiera wszystkie słowa występujące w ich treści2. System tworzy z zebranych słów słownik ustawiając je na ustalonych miejscach
Przepływ alternatywny:	Brak.
Warunki końcowe	Utworzony słownik

Tablica 3.5. Scenariusz dla przypadku użycia: Utwórz słownik

3.2.6. Zbuduj reprezentację Bag of Words

Nr przypadku użycia:	6
Nazwa przypadku użycia:	Zbuduj reprezentację Bag of Words
Aktorzy:	System
Wymagania wstępne:	Zbudowany słownik, witryny zaktualizowane o treść
Przepływ:	<ol style="list-style-type: none">1. System pobiera dane o zawartości witryn z bazy danych2. System na podstawie słów w słowniku, zlicza ilość wystąpień danych słów na poszczególnej witrynie3. System z liczby wystąpień wszystkich słów słownika tworzy wektor dla każdej strony4. System aktualizuje w bazie danych witryny o utworzone wektory
Przepływ alternatywny:	<ol style="list-style-type: none">1a. System nie może połączyć się z bazą danych4a. System nie może połączyć się z bazą danych
Warunki końcowe	Witryny zaktualizowane o wektory

Tablica 3.6. Scenariusz dla przypadku użycia: Zbuduj reprezentację Bag of Words.

3.2.7. Zbierz podstrony

Nr przypadku użycia:	7
Nazwa przypadku użycia:	Zbierz podstrony
Aktorzy:	System
Wymagania wstępne:	Poprawny adres witryny url w bazie
Przepływ:	<ol style="list-style-type: none">1. System pobiera adres witryny2. System przeszukuje znajdujące się na niej podstrony3. System pobiera zawartość przeszukanych podstron
Przepływ alternatywny:	Brak
Warunki końcowe	Podstrony dodane do bazy danych

Tablica 3.7. Scenariusz dla przypadku użycia: Zbierz podstrony.

3.2.8. Trenuj model SVM

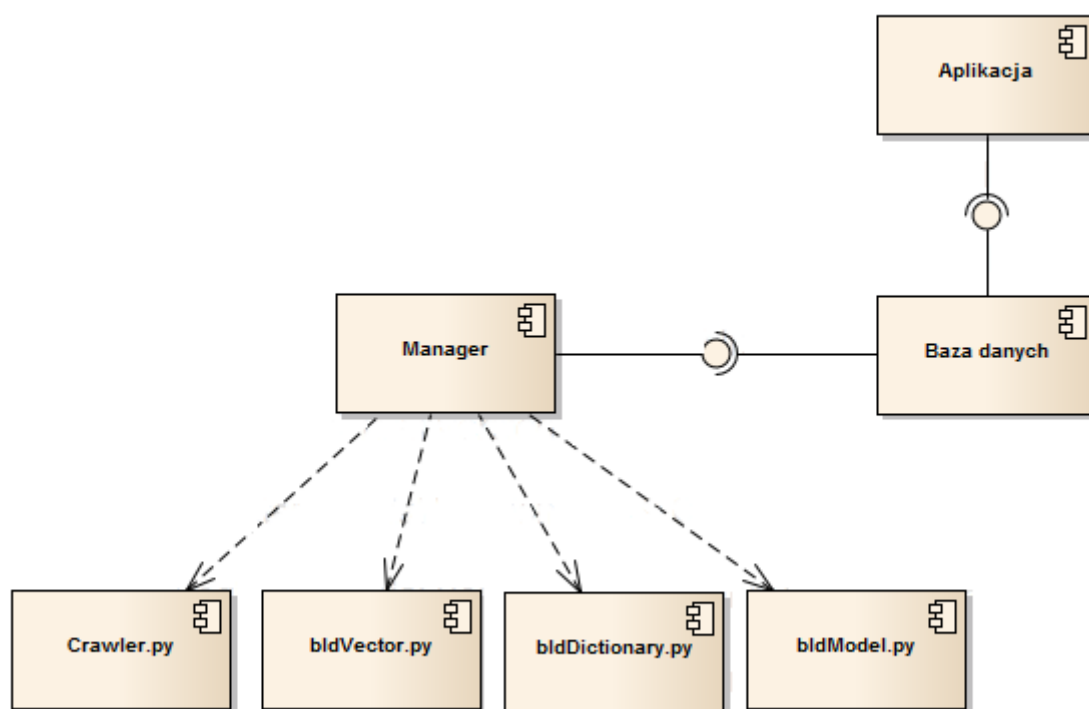
Nr przypadku użycia:	8
Nazwa przypadku użycia:	Trenuj model SVM
Aktorzy:	System
Wymagania wstępne:	Zbudowana reprezentacja wektorów dla witryn.
Przepływ:	<ol style="list-style-type: none">1. System pobiera wektory witryn i nadane im kategorie2. System umieszcza dane na dwóch oddzielnych listach3. System uruchamia algorytm SVM na pobranych danych4. System buduje model służący do kategoryzowania przyszłych stron na podstawie już określonych
Przepływ alternatywny:	1a. System nie może się połączyć z bazą danych
Warunki końcowe	Zbudowany model SVM

Tablica 3.8. Scenariusz dla przypadku użycia: Trenuj mode SVM

3.2.9. Nadaj kategorię dla strony

Nr przypadku użycia:	9
Nazwa przypadku użycia:	Nadaj kategorię dla strony
Aktorzy:	System
Wymagania wstępne:	Utworzony model SVM, witryny w bazie danych bez nadanej kategorii
Przepływ:	<ol style="list-style-type: none">1. System pobiera wszystkie strony które nie otrzymały przydzielonej kategorii2. Na podstawie uprzednio utworzonego modelu SVM nadaje kategorie dla każdej strony3. System zapisuje do bazy danych skategoryzowane adresy stron
Przepływ alternatywny:	1a. System nie może połączyć się z bazą danych.
Warunki końcowe	Wszystkie strony posiadają określoną kategorię.

Tablica 3.9. Scenariusz dla przypadku użycia: Nadaj kategorię dla strony.



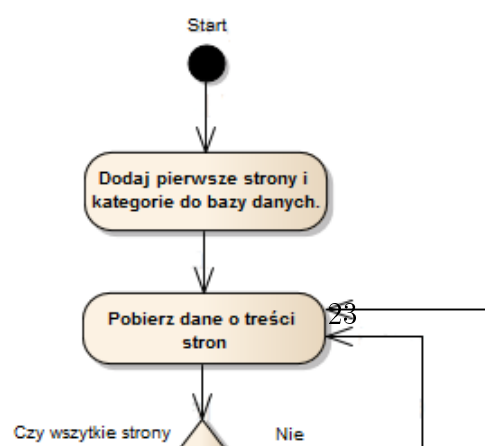
Rysunek 4.1. Diagram komponentów

Rozdział 4

Implementacja aplikacji

4.1. Architektura aplikacji - diagram komponentów

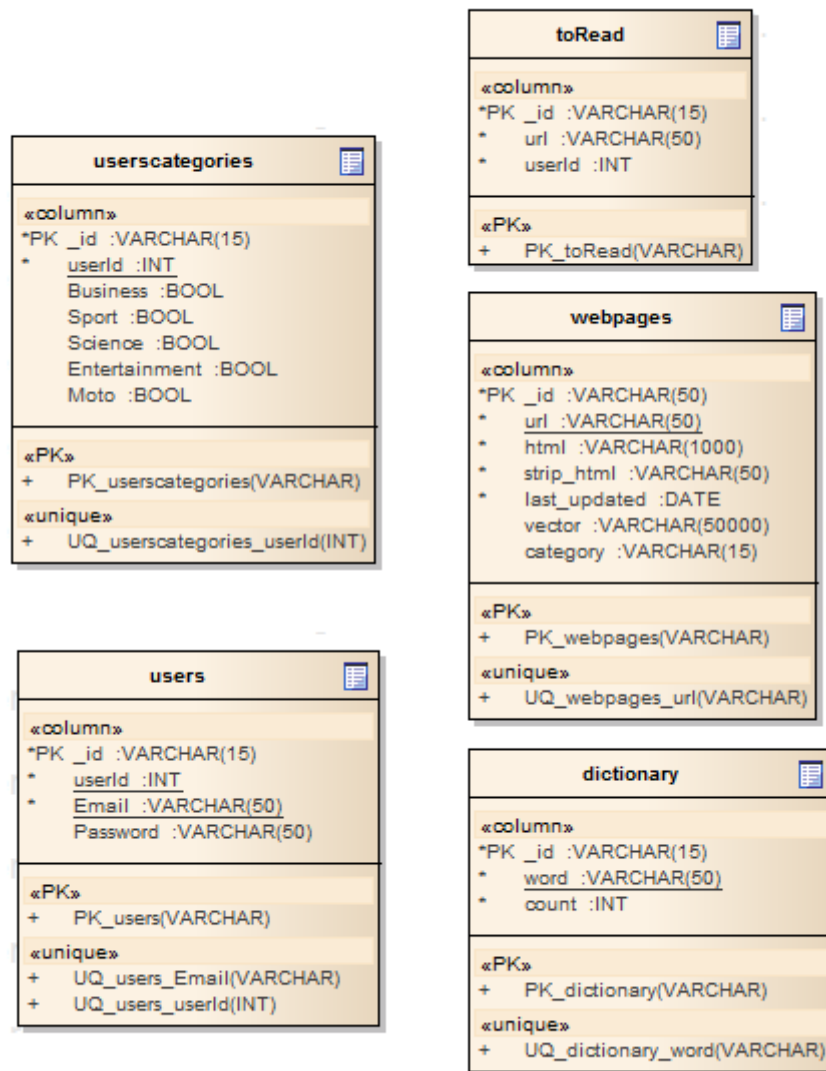
4.2. Sposób działania - diagram czynności



Przy pierwszym uruchomieniu aplikacji uruchomia się najpierw **Crawler WWW**, który będzie zbierał kody źródłowe ze strony podanej przez nas, stron z baz danych oraz z ich podstron. Po zebraniu odpowiedniej ilości danych uruchomia się skrypt tworzenia słownika, gdzie na podstawie znalezionych przez nas stron, ułoży on słownik złożony ze wszystkich słów które zostały zebrane oraz zmierzy ich liczebność, zamieniając potem tekst na wektory metodą **Bag of words**. Następnie po określeniu kategorii na podstawie pobranych stron, możemy przepuścić dowolne strony przez klasyfikator **SVM**, który określi przydział strony do kategorii. Skategoryzowane oraz zweryfikowane strony są przekazywane do wyświetlania użytkownikowi. Zalogowany użytkownik określa czy dana kategorii mu odpowiada, odznaczając nazwę kategorii w opcjach swojego konta, czy nie chciałby w przyszłości oglądać tego typu wiadomości, nie zaznaczając pola z odpowiadającą kategorią. Użytkownik może sam dodawać nowe strony, które weryfikuje w tle pracujący **Crawler WWW** oraz klasyfikator **SVM**. W aplikacji jest zarejestrowany zawsze jeden moderator, który przy przeglądaniu stron, w razie gdyby wyświetlana strona została źle przydzielona do kategorii, może sam ustawić poprawną kategorię dla wyświetlanej strony poprzez wpisanie jej w wyświetlające się okno. Wszystkie skrypty zostały wykonane w języku **Python**.

4.3. Schemat bazy danych

Baza danych jest reprezentowana przez nierelacyjną bazę MongoDB.



Rysunek 4.3. Schemat bazy danych

W bazie występują następujące tabele:

users - Tabela zawierająca dane o użytkownikach zarejestrowanych w serwisie

webpages - Tabela w której zamieszczone są strony z informacjami pobrane i wcześniej zweryfikowane przez system kategoryzujący. Pole url zawiera adres danej strony, html - jej całą treść wraz z tagami html i funkcjami, strip_html - treść oczyszczoną ze znaczników, last_updated - ostatnią datę aktualizacji, vector - wektor zbudowany metodą Bag of Words, category - określoną kategorię danej strony.

dictionary - Słownik, na podstawie którego algorytm kategoryzuje strony. Zawiera słowa oraz ich ilość wystąpień w dokumentach.

toRead - Tabela która przechowuje dane o stronach do wyświetlenia dla poszczególnych użytkowników. Przechowuje id użytkownika i adres strony do wyświetlenia.

userscategories - Tabela zawierająca informacje o wybranych kategoriach przez użytkowników

4.4. Implementacje

W tej sekcji zamieszczono kody źródłowe do skryptów wykonanych na potrzeby aplikacji. Wszystkie skrypty zostały utworzone w języku Python.

4.4.1. Implementacja Crawlera WWW

Crawler pobiera adres witryny z bazy danych, a następnie pobiera jej zawartość oraz zawartość jej podstron. Sprawdza czy znajdują się już one w danej sesji, nadaje im pełne adresy i pobiera do zmiennych. Zawartość strony jest zapisywana i oczyszczana z całego kodu JavaScript i tagów HTML. Po tym program zapisuje stronę z danymi do bazy danych.

```
1 now = datetime.datetime.now()
2 client=MongoClient('mongodb://127.0.0.1:3001/meteor')
3 db=client.meteor
4 webpages=db.webpages
5 cleaner = Cleaner()
6 cleaner.javascript = True
7 cleaner.style = True
8 def remove_html_markup(s):
9     tag = False
10    quote = False
11    out = ""
12    for c in s:
13        if c == '<' and not quote:
14            tag = True
15        elif c == '>' and not quote:
16            tag = False
17        elif (c == '"' or c == "'") and tag:
18            quote = not quote
19        elif not tag:
20            out = out + c
21    return out
22
23 class Crawler:
24     def Run(self):
25         print "Crawling..."
26         client=self.CreateConnection()
27         db=client.meteor
28         webpages=db.webpages.find()
29         for webpage in webpages:
30             myurl = webpage.get('url')
31             self.Crawl(myurl,db)
32
33     def CreateConnection(self):
34         try:
35             return MongoClient('mongodb://127.0.0.1:3001/meteor')
36         except:
37             print "ERROR: _Can't_connect_to_database."
38
39
40 def Crawl(self,myurl,db):
41     now = datetime.datetime.now()
42     urllist = []
43     for i in re.findall('\"href=[\"'](.\"'+)
```

```

44         urllib.urlopen(myurl).read(), re.I):
45     if i.find("http://") == -1 and i.find("https://") == -1:
46         i=myurl+i;
47 try:
48     for ee in re.findall('''href=["'](.["']+)["']''',
49         urllib.urlopen(i).read(), re.I):
50         if ee.find("http://") == -1 and ee.find("https://") == -1:
51             ee=myurl+ee
52             if (ee in urllist) == False:
53                 urllist.append(ee)
54                 response = urllib2.urlopen(ee)
55                 page_source = response.read()
56                 page_source = '_'.join(page_source.split())
57                 page_source_strip = cleaner.clean_html(page_source)
58                 page_source_strip = remove_html_markup(page_source_strip)
59                 page_source_strip = ''.join(c for c in page_source_strip
60                     if c not in ',: "1234567890().[]{};!@#$$%^&*-_+=/?')
61                 webpage = {"url" : ee, "html_strip": page_source_strip,
62                     "last_update":now, "status": "Created", "vector": []}
63                 try:
64                     db.webpages.insert_one(webpage)
65             except:
66                 print "Can't insert webpage into database."
67 except Exception:
68     pass

```

Listing 4.1. Kod źródłowy Crawlera WWW

4.4.2. Implementacja tworzenia słownika

Program zlicza unikalne słowa z pobranych wcześniej przez Crawlera stron www. Program zbiera i zlicza wszystkie słowa. Następnie metodą **Bag of words** zamienia je na jeden wektor.

```
1 class DicBuilder:
2     def CreateConnection(self):
3         try:
4             return MongoClient('mongodb://127.0.0.1:3001/meteor')
5         except:
6             print "ERROR: _Can't_connect_to_database."
7
8     def CountWords(self, webpages, db):
9         wordcount={}
10        for wp in webpages:
11            for word in wp.get('html_strip').split():
12                if word not in wordcount:
13                    wordcount[word] = 1
14                else:
15                    wordcount[word] +=1
16            try:
17                db.webpages.update_one({ "_id": wp.get("_id") },
18                                       { "$set": { "status": "Scanned" } })
19            except:
20                print "ERROR: _Can't_update_webpage_in_database."
21        return wordcount
22
23    def UpdateDictionaryToDB(self, wordcount, db):
24        for k,v in wordcount.items():
25            item = {"word" : k, "count" : v}
26            try:
27                db.dictionary.insert_one(item)
28            except:
29                print "ERROR: _Can't_insert_word_into_dictionary."
30        print len(wordcount)
31
32    def Build(self):
33        print "Creating_Dictionary..."
34        client=self.CreateConnection()
35        db=client.meteor
36        webpages=db.webpages.find()
37        wordcount = self.CountWords(webpages,db)
38        self.UpdateDictionaryToDB(wordcount,db)
39        print "Dictionary_Created."
```

Listing 4.2. Kod źródłowy tworzenia słownika

4.4.3. Implementacja tworzenia wektorów

Poniższy skrypt pobiera dane o stronie z tabeli **webpages**. Zlicza słowa kluczowe określone wcześniej w słowniku i je zlicza, następnie na podstawie liczby wystąpień słów tworzy wektor, który zapisuje w tej samej tabeli przy analizowanej stronie.

```
1 class VecBuilder:
2     def Build(self):
3         print "Building_Vectors"
4         client=self.CreateConnection()
5         db=client.meteor
6         try:
7             cursor_d = db.dictionary.find()
8         except:
9             print "Can't_download_dictionary_from_database."
10        try:
11            cursor_wp = db.webpages.find()
12        except:
13            print "Can't_download_webpages_from_database."
14        dictionary = self.GetWords(cursor_d)
15        self.BuildVectors(cursor_wp, db, dictionary)
16        print "Vectors_updated."
17
18    def CreateConnection(self):
19        try:
20            return MongoClient('mongodb://127.0.0.1:3001/meteor')
21        except:
22            print "ERROR: Can't_connect_to_database."
23
24    def GetWords(self, cursor_d):
25        dictionary = []
26        for obj in cursor_d:
27            dictionary.append(obj.get('word').encode("utf8"))
28        return dictionary
29
30    def BuildVectors(self, cursor_wp, db, dictionary):
31        vector = []
32        for webpage in cursor_wp:
33            if not webpage.get('vector'):
34                del vector[:]
35                vector[:] = []
36                cursor_d = db.dictionary.find()
37                text = webpage.get('html_strip').encode("utf8")
38                for obj in dictionary:
39                    vector.append(text.count(obj))
40                db.webpages.update_one({"_id": webpage.get("_id")},
41                                       {'$push':{'vector':{'$each':vector}}})
```

Listing 4.3. Kod źródłowy tworzenia wektorów

4.4.4. Implementacja budowania modelu SVM

Program pobiera kategorie i wektory stron które posiadają określoną już kategorię, następnie tworzy z nich model SVM.

```
1 class BuildModel:
2     def Build(self):
3         print "Buiding_Model"
4         client=self.CreateConnection()
5         db=client.meteor
6         try:
7             cursor_wp = db.webpages.find()
8         except:
9             print "Can't_download_webpages_from_database."
10        X = []
11        y = []
12        for webpage in cursor_wp:
13            if webpage.get('category'):
14                X.append(webpage.get('vector'))
15                y.append(webpage.get('category'))
16        try:
17            clf = svm.SVC()
18            clf.fit(X,y)
19        except:
20            print "Can't_build_a_model."
21    def PredictCategory(vector, self):
22        return clf.predict(vector)
23    def CreateConnection(self):
24        try:
25            return MongoClient('mongodb://127.0.0.1:3001/meteor')
26        except:
27            print "ERROR: Can't_connect_to_database."
```

Listing 4.4. Kod źródłowy budowania modelu SVM

4.4.5. Implementacja menadżera

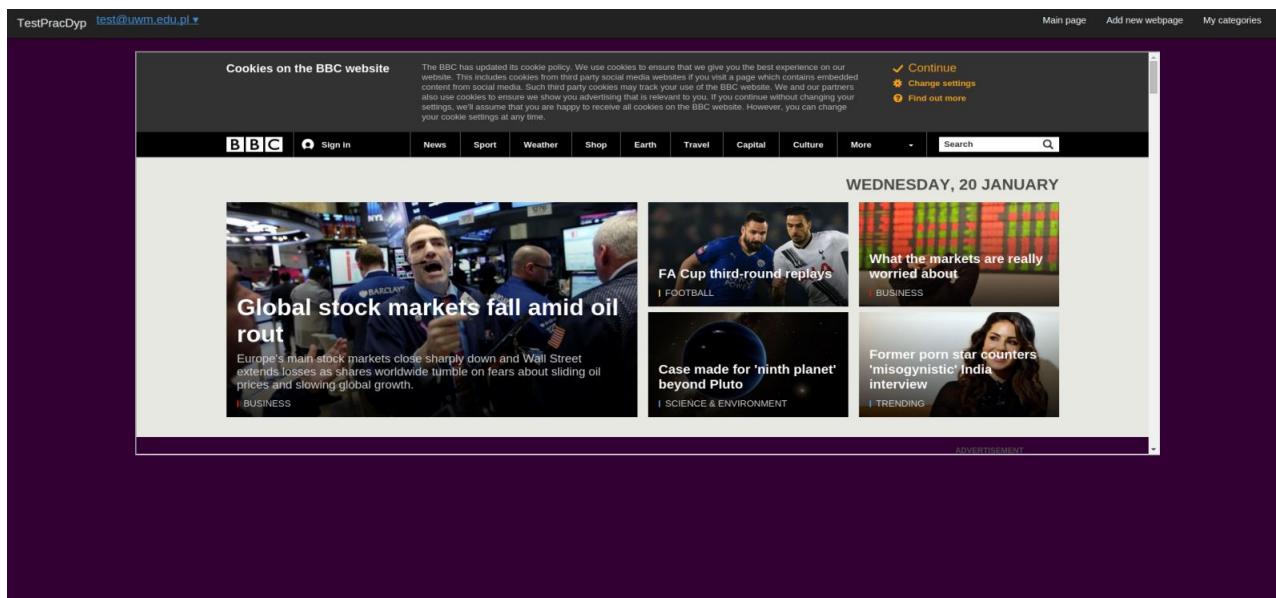
Skrypt pobiera klasy ze wszystkich poprzednich skryptów i uruchamia je w określonej kolejności. Po utworzeniu wektorów, słownika i modelu, określa kategorię dla witryn, które jeszcze jej nie posiadają.

```
1 from crawler import Crawler
2 from bldvector import VecBuilder
3 from bldmodel import BuildModel
4 from dictionary import DicBuilder
5 from pymongo import MongoClient
6
7 client=MongoClient('mongodb://127.0.0.1:3001/meteor')
8 db=client.meteor
9
10 Dictionary = DicBuilder()
11 crawler = Crawler()
12 vectorBuilder = VecBuilder()
13 bldModel = BuildModel()
14
15 crawler.Run()
16 Dictionary.Build()
17 vectorBuilder.Build()
18 bldModel.Build()
19 cursor_wp = db.webpages.find()
20
21 for webpage in cursor_wp:
22     if not webpage.get('category'):
23         category = bldModel.PredictCategory(webpage.get('vector'))
24         db.webpages.update_one({ "_id": webpage.get("_id") },
25                                { '$push': { 'category' : category} })
```

Listing 4.5. Kod źródłowy menadżera

4.5. Interfejs użytkownika

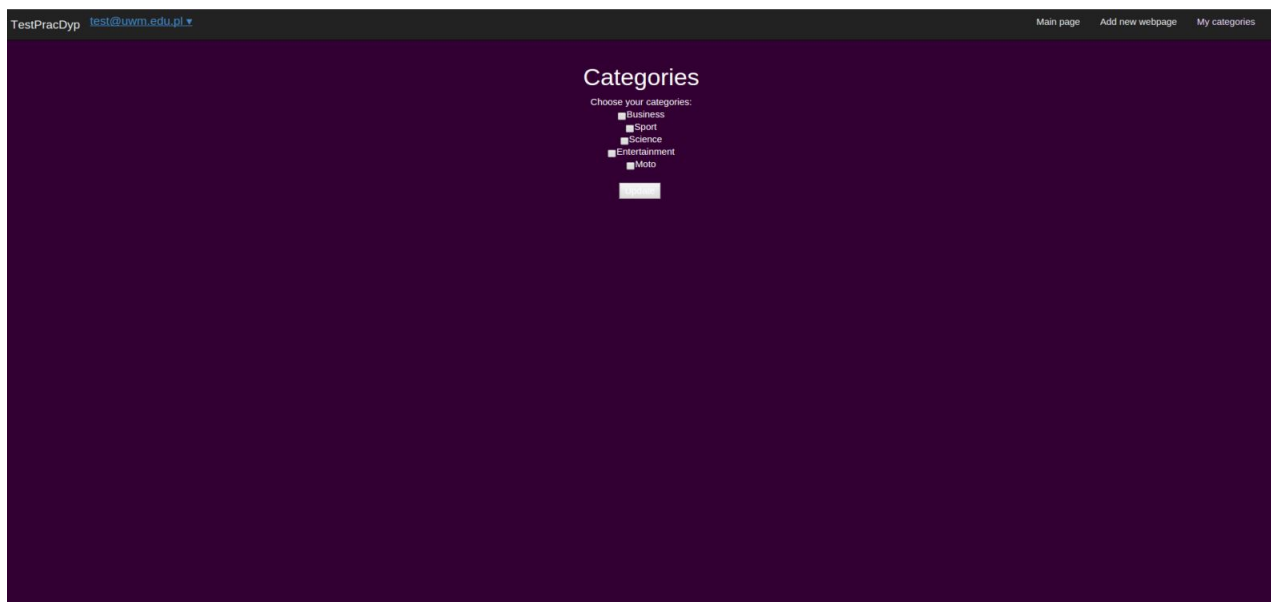
4.5.1. Strona główna



Rysunek 4.4. Strona główna

Największą część głównej strony zajmuje okno do wyświetlania wiadomości dla użytkownika. Pod artykułem mamy przycisk, który pozwala przejść do kolejnego artykułu. Z kolei u góry mamy pasek z opcjami do wyboru, który będzie pokazywał się na każdej podstronie.

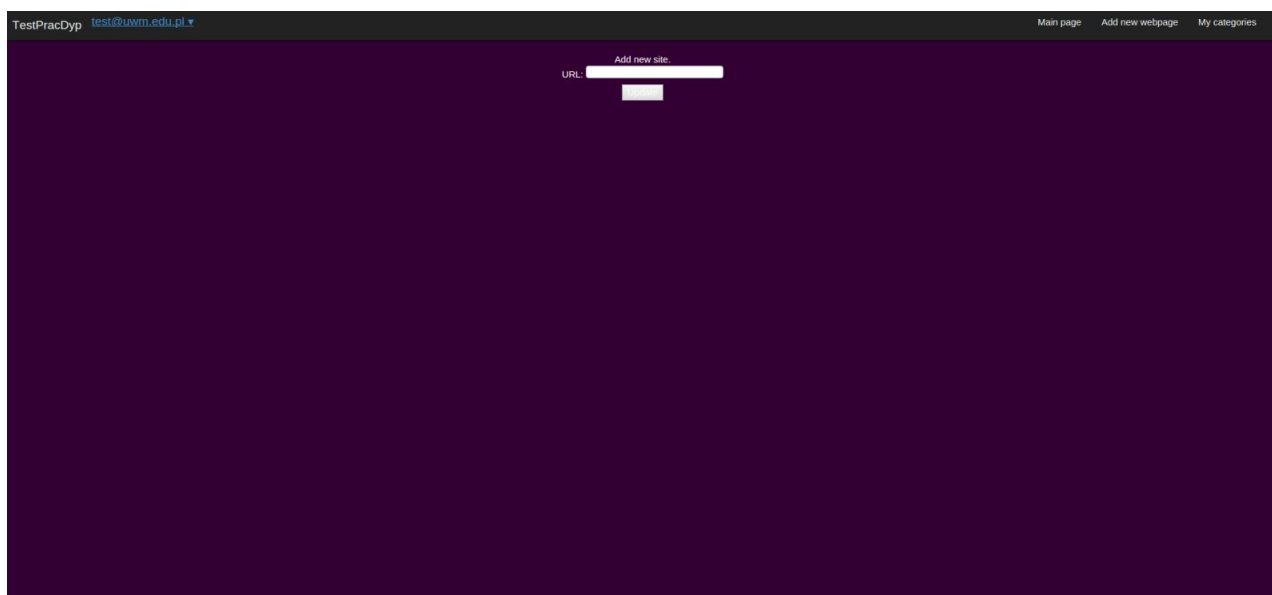
4.5.2. Kategorie



Rysunek 4.5. Kategorie

W tej części aplikacji możemy przeglądać swoje kategorie, a także nimi zarządzać.

4.5.3. Dodaj stronę



The screenshot shows a web application interface with a dark purple background. At the top, there is a navigation bar with the text 'TestPracDyp' and a dropdown menu 'test@uwm.edu.pl'. To the right of the navigation bar are three links: 'Main page', 'Add new webpage', and 'My categories'. In the center of the page, there is a form titled 'Add new site.' with a label 'URL:' followed by a text input field and a submit button.

Rysunek 4.6. Dodaj stronę

Na tej stronie użytkownik ma możliwość dodania własnej strony, która będzie dodana do bazy danych strony i będzie otrzymywał z niej wiadomości.

4.6. Instrukcja instalacji

Aplikacja została stworzona na systemie operacyjnym Linux ubuntu 14.04 LTS i zaleca się emulację aplikacji na tym systemie.

4.6.1. Instalacja Node.js

Meteor bazuje na środowisku Node.js, aby je zainstalować należy przejść do strony <http://nodejs.org> i kliknąć w przycisk „INSTALL”. Po tej procedurze rozpocznie się pobieranie i instalacja node.js.

4.6.2. Instalacja Meteora

Po zainstalowaniu Node.js, można przystąpić do instalacji środowiska Meteor. Aby zainstalować Meteor'a należy otworzyć terminal systemu i wpisać w nim następującą komendę:

```
curl https://install.meteor.com | /bin/sh
```

Dodatkowo należy zainstalować moduł Meteorite do zarządzania systemem wersji Meteor'a i jego modułami. Aby tego dokonać należy wpisać w terminalu:

```
sudo -H npm install -g meteorite
```

4.6.3. Uruchamianie aplikacji

Folder z aplikacją należy umieścić na dysku twardym komputera. Następnie należy uruchomić terminal, poprzez który dostajemy się do folderu z aplikacją i wpisujemy komendę:

```
meteor
```

Pozwoli nam to na uruchomienie aplikacji na localhoście. Aby przeglądać zawartość bazy danych, należy przy uruchomionej aplikacji otworzyć drugie okno terminala i wpisać w nim komendę:

```
meteor mongo
```

Rozdział 5

Podsumowanie

Bibliografia

- [1] ECMAScript - Standard ECMA-262
<http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- [2] Atmosphere - Meteor Package Manager
<https://atmospherejs.com/>
- [3] scikit-learn 0.17.0 - Machine Learning in Python
<http://scikit-learn.org/stable/>
- [4] ASP.Net MVC
<http://www.asp.net/mvc>
- [5] Meteor 1.1: now supporting Microsoft Windows and MongoDB 3.0
<http://info.meteor.com/blog/meteor-1.1-microsoft-windows-mongodb-3.0>
- [6] Coleman T., Greif S.: *Discover Meteor - Building real-time javascript web apps*, 2014
- [7] Wrycza S., Marcinkowski B., Wyrzykowski K.: *Język UML 2.0 w modelowaniu systemów informacyjnych*, Helion, 2006

Spis rysunków

1.1.	Liczba stron internetowych na przestrzeni lat.	5
2.1.	Struktura danych Meteor'a.	7
2.2.	Przepływ danych w środowisku Meteor.	9
2.3.	Przykładowy rekord w MongoDB	11
2.4.	Przykład granicy decyzyjnej wyznaczonej przez SVM.	12
2.5.	Typy jąder SVM	13
2.6.	Porównanie dwóch wektorów utworzonych metodą Bag of words	14
3.1.	Diagram przypadków użycia	15
4.1.	Diagram komponentów	23
4.2.	Diagram czynności	23
4.3.	Schemat bazy danych	25
4.4.	Strona główna	33
4.5.	Kategorie	34
4.6.	Dodaj strone	35

Listings

4.1	Kod źródłowy Crawlera WWW	27
4.2	Kod źródłowy tworzenia słownika	29
4.3	Kod źródłowy tworzenia wektorów	30
4.4	Kod źródłowy budowania modelu SVM	31
4.5	Kod źródłowy menadżera	32